Geoinformatica (2007) 11:331–357
DOI 10.1007/s10707-006-0003-y

Unified Hybrid Terrain Representation Based on Local Convexifications

M. Bóo · M. Amor · J. Döllner

Received: 13 July 2005 / Revised: 6 June 2006 /
Accepted: 26 June 2006 / Published online: 1 February 2007

Abstract Hybrid digital terrain models represent an effective framework to combine and integrate terrain data with different topology and resolution. Cartographic digital terrain models typically are constituted by regular grid data and can be refined by adding locally TINs that represent morphologically complex terrain parts. Direct rendering of both data sets to visualize the digital terrain model would generate geometric discontinuities as the meshes are disconnected. In this paper we present a new meshing scheme for hybrid terrain representations. High quality models without discontinuities are generated as the different representations are softly joined through an adaptive tessellation procedure. Due to the complexity of the algorithms involved in the tessellation procedure, we propose a mixed strategy where part of the information is pre-computed and efficiently encoded. This way, for rendering the model, the tessellation information has to be decoded and only additional simple operations have to be performed.

Keywords Hybrid terrain representation · Convexification · Interactive visualization

M. Bóo (✉)
Department of Electronic and Computer Engineering,
University of Santiago de Compostela,
Santiago de Compostela, Spain
e-mail: mboo@dec.usc.es

M. Amor
Department of Electronics and Systems, University of A Coruña,
A Coruña, Spain
e-mail: margamor@udc.es

J. Döllner
Hasso–Plattner-Institut, University of Potsdam,
Potsdam, Germany
e-mail: doellner@hpi.uni-potsdam.de

1 Introduction

An increasing number of applications and systems apply and provide interactive 3D geovisualization such as in the fields of Geographic Information Systems (GIS), urban and landscape planning, network planning in telecommunication and energy supply, disaster management and visual spatial analytics, and cartography. Digital terrain models (DTM) constitute fundamental elements of the underlying 3D geovirtual environments and models in these application domains. For large-scale terrain models, the storage, transmission, and processing requirements demand for effective encoding and rendering techniques to cope with their geometric complexity and to facilitate real-time rendering.

The representation type of a DTM depends on several factors, such as the nature of the capturing technology, input data, application domain, and computational resources available. High resolution and geometric detail commonly requires high storage costs and computation time [9]–[11]. Therefore, to be effective, the model should maintain a good trade-off between resolution and storage requirements. Applications, in addition, can demand on specific topological and geometrical features and operations of the DTM.

A simple method for approximating terrain surfaces uses a regular grid Digital Elevation Model (DEM), in which a set of sampled points representing measures of altitude or elevation are stored at regular intervals [17]. A disadvantage of the DEM is its inherent spatial invariability, since the structure is not adaptive to the irregularity of the terrain. This may produce a large amount of data redundancy, especially where the topographic information is minimal. Alternatively, a Triangulated Irregular Network (TIN) model approximates a topographic surface by a set of non-overlapping contiguous triangular faces generated from a finite set of sampled data points [17]. In this representation the terrain data is commonly irregularly distributed in space.

Real-world terrain data sets are composed of different types of data. For example, a cartographic terrain surface model can include a grid data set, describing the terrain surface over a wider area, and a TIN data set, describing terrain structures of finer resolution such as topographically complex terrain features or man-made terrain structures and objects. In particular, microstructures allow us to enhance existing grid-based DEMs by adding details to specific regions of interest without having to increase the grid resolution or having to convert the whole terrain model into a TIN-based representation. Therefore, hybrid terrain models represent a memory-efficient approach for detailing complex terrain models. Hybrid terrain models, in addition, facilitate the implementation of 3D terrain editing techniques, which require the ability to locally replace and refine the terrain surface model. In the following, we assume that a DEM is represented by a grid-base data set over the whole area, while only small portions of the terrain surface are refined by TIN-based data sets. Morphologically important terrain features (e.g., rivers, streets, and ridges) typically affect approximately less than 20% of the overall terrain surface. Furthermore, microstructures integrated in grid-based terrain models improve the graphical and perceptual quality of visualized terrain models since they provide a more precise and adequate geometry, shading, and illumination of morphologically important terrain features. In interactive 3D visualization, microstructures play a key role to improve the level-of-detail management. For example, far-away views to the terrain

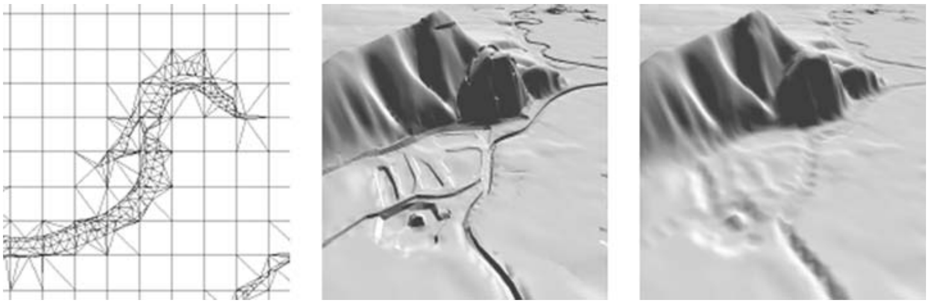


Fig. 1 Hybrid terrain model specified by a reference grid and partially refined TINs (*left*), view of the terrain model with TINs (*center*) and without TINs (*right*)

can omit microstructures, whereas close-by views can seamlessly integrate them. The presented unified hybrid terrain approach provides such integrated representation of coarse-grained grid-data and fine-grained TIN data.

As both kind of terrain data sets, grid and TINs, are typically obtained from different sources (based on, for example, different capturing techniques or measurement systems), these meshes are not directly connected. As suggested in [3] and in order to avoid discontinuities in the junction between representations, both meshes have to be locally adapted. This can be performed through an adaptive tessellation of the grid cells in the neighbourhood of the TIN border. As an example see Fig. 1 where the grid cells in the neighbourhood of the TIN structure are adaptively tessellated so that no discontinuities are generated.

Including multiresolution in the grid representation implies that the adaptive tessellation would depend on the level of detail (LOD). Due to this dependence, pre-computing and transmitting LOD-dependent information would not be efficient. The desired framework would be developing an efficient representation of all LOD tessellations. This unified representation would be sent to the graphics processing unit (GPU) only once, where the specific tessellations for the required LOD would be generated. Note that LOD representations of high-resolution grid-based terrain surface models do not represent an alternative to hybrid mesh representation: Topologically and geometrically TINs are more precise in representing specific terrain features, including natural terrain feature (e.g., ridges and edges) and man-made features (e.g., embankments and street slopes). Grids, in general, can only approximate these features while the memory requirement drastically grows if the resolution increases. TIN microstructures for local refinements together with regular grid data for the overall terrain representation prove to be optimal with respect to memory requirements and precision.

Although terrain interactive visualization and the utilization of multiresolution techniques for grids and for TINS are very active fields of research [5], [11], [12], [14], [15], we have no found any proposal in the bibliography that develops a specific solution for hybrid terrain models.

In this paper we present a new meshing algorithm for hybrid representation of terrains combining multiresolution grids with high resolution TINs. The method we propose permits an efficient representation of the different LOD tessellations employed for adapting the meshes in a unified representation. In this way, a unique set of information permits storage/generation of the different tessellations corresponding to all possible levels of detail in an efficient way. The GPU receives the information and once a specific LOD is selected, the tessellation for adapting the TIN and the grid can be simply and efficiently decoded. As this representation is generic and independent of the view point, the information has only to be sent once from the CPU to the GPU.

The information is employed by the GPU to generate the adaptive tessellation required to connect the grid and the TIN. We propose an adaptive tessellation algorithm based on the connection between the cell corners and the TIN border previously convexified. The representation we propose permits, with small modifications, the generation of two different local adaptive tessellations, one based on triangles while the other combines triangles and sub-cells. While the first one generates fewer primitives for rendering, the second reduces the tessellation area to a close neighbourhood of the TIN. As both tessellations are based on the same representation procedure and share common resources we present a combined solution that permits both solutions to coexist. Moreover, as quality results and processing times depend on the specific application, the user could choose one of these two tessellation modes depending on the specific requirements.

2 Hybrid Meshing Algorithm

The hybrid representation of terrains combining regular grids and TINs, requires the adaptive tessellation of the grid cells in the neighbourhood of the TIN. Taking into account that the regular grid can be represented with different resolutions, computing and sending the tessellations required for a specific LOD would not be efficient. In the solution we propose, a unified representation of all LOD-dependent tessellations is generated. This information is employed in the GPU to decode the tessellation required for a specific LOD.

This is schematically depicted in Fig. 2. The TIN and grid meshes are sent to the graphics pipeline together with the unified tessellation representation. As most of the tessellation information is generic and independent on the view point, this information has to be sent only once. The primitives associated with the adaptive tessellation are decoded and generated in GPU.

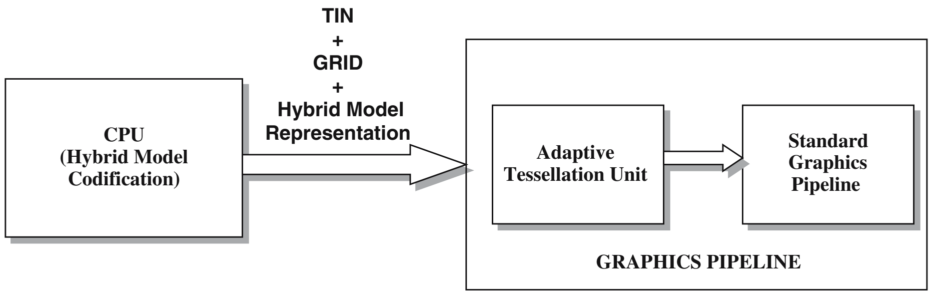

TIN
+
GRID
+
Hybrid Model Representation
CPU (Hybrid Model Codification)
Adaptive Tessellation Unit
Standard Graphics Pipeline
GRAPHICS PIPELINE

Fig. 2 Generic structure of the system

With real-time rendering as an objective, a specific hardware unit could be included for implementing these algorithms. Similar techniques were proposed for adaptive tessellation in other frameworks, such as, for example, adaptive tessellation of triangle meshes based on displacement map information [2], [4], [6], where the adaptive tessellations are performed in specific hardware for the application. As required characteristics for such a strategy, the algorithm to be implemented should be simple.

In the following, we present the generic structure of the adaptive tessellation method we propose. In our algorithm the TIN together with the grid cells that are *not covered* (*NC* cells) by the TIN can be directly rendered, the grid cells *completely covered* (*CC* cells) are eliminated and only those grid cells *partially covered* (*PC* cells) by the TIN have to be adaptive tessellated. As previously mentioned, the methodology we present for tessellation representation can be employed, with small modifications, for generating two different kinds of tessellation schemes. In the following we will focus the presentation on one of the methods in which only triangles are generated while the other, in which triangles and sub-cells are produced, is presented as an extension in Section 5.

As will be shown later, working with TIN structures locally convex in each *PC* cell ensures an easy tessellation procedure. The simplicity and regularity of the tessellation procedure can be implemented in hardware following the general framework indicated in Fig. 2.

The strategy we propose is based on local convexification of the TIN boundary inside each cell. Convexification of triangle meshes is a complex and irregular algorithm. In our proposal, the convexification for the different levels of detail are computed and encoded in a compact and efficient way as a pre-processing step in the CPU. This is possible because the convexification is performed incrementally from the finest to the coarsest level of detail associated with the grid. This strategy permits the generation of a unified representation of all convexification levels and, on the other hand, permits the generation of coherent transitions between contiguous cells corresponding to different LODs. This information is efficiently encoded and sent to the graphics card, where the convexification triangles corresponding to a specific LOD are decoded and rendered.

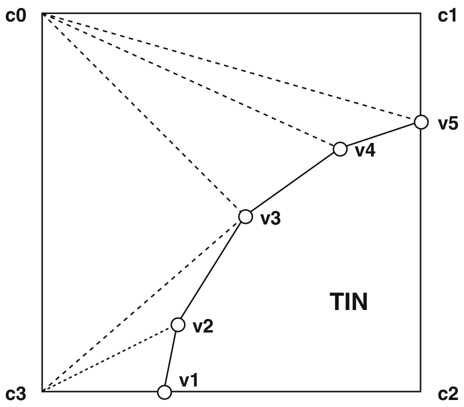
2.1 Cell Tessellation based on Convex TIN Structures: Corner Tessellation

The partially covered cells are tessellated to connect them with the TIN. In the procedure we present, uncovered corners are identified and consecutively connected with the TIN boundary vertices. In this subsection we present the corner tessellation algorithm considering that the TIN has been previously convexified.

One of the keys of the algorithm is determining when a *shift* in the grid corner should be performed i.e., when a new corner has to be employed instead of the current one. This can be performed by the evaluation of the angles of edges between the corner and the consecutive vertices of the TIN and specifically, by analyzing the sign of the cross product of consecutive edges.

Consider the example depicted in Fig. 3 where a *PC* cell and the corresponding convex TIN silhouette are depicted. Let us assume that the TIN boundary vertices (v_1, v_2, v_3, v_4 and v_5 in this example) and the cell corners uncovered by the TIN (c_3, c_0, c_1) are processed in a clockwise order. In the procedure we propose the list of

Fig. 3 Corner tessellation example



corners (c_3, c_0, c_1) is connected with the list of vertices (v_1, v_2, v_3, v_4, v_5) following a sequential order. This way, corner c_3 is consecutively connected to the vertices of the boundary while possible. In this example corner c_3 is connected with vertex v_1, v_2 and v_3, but not with v_4 because the new triangle (c_3, v_3, v_4) would overlap with the TIN. Then, corner c_0 is selected and also connected with v_4 and with v_5.

The simple tessellation procedure is based on the TIN convex structure. Next we present the incremental convexification procedure we propose. As mentioned before, the incremental strategy permits a unified representation of the different tessellation levels.

2.2 Incremental Convexification

In the algorithm we propose, the TIN boundary is convexified to allow an easy tessellation between the convex boundary and the cell corners. The convexification inside each cell has three steps: First, the convex hull of the TIN boundary is computed. After this, all cavities are identified and finally triangulated. Similar procedures were previously proposed in the context of non-convex tetrahedral mesh visualization [7], [8], [16].

In the following and to simplify the presentation, we assume that there is a vertex in each intersection point between TIN boundary and each cell border of the finest level of detail. In a generic case, these intersection vertices would have to be computed for a correct convexification procedure. A detailed analysis of the intersection vertices computation can be found in [1].

In the first step the convexification is performed for the finest grid resolution level. Figure 4a shows four cells corresponding to the finest resolution level of the grid and the corresponding TIN silhouette covering this area. The TIN is depicted in grey and the TIN boundary is explicitly marked and vertices on it are indicated with circles. The local convexification and triangulation results are indicated in Fig. 4b. Local convex hulls are delimited by vertices $\{0, 1, 3, 4\}$ (up-right cell), $\{4, 12\}$ (up-left cell) and $\{12, 13, 15, 16\}$ (down-left cell). Once the convex hull inside each cell is determined, any standard tessellation algorithm could be applied for generating triangles inside the caves [13].

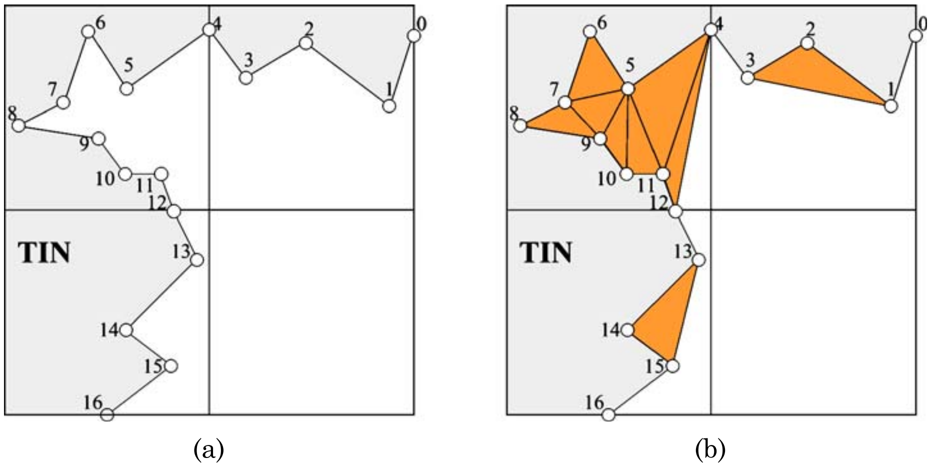


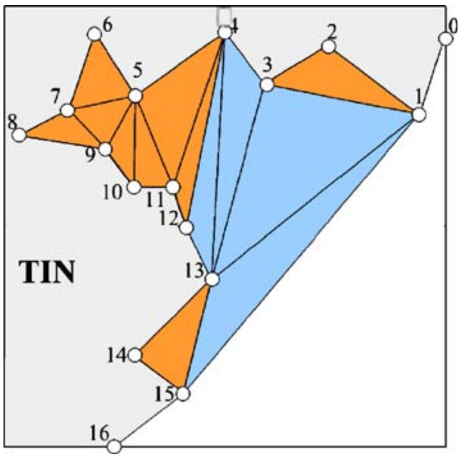
Fig. 4 **a** Four cells of the grid with the TIN silhouette **b** Local convexification

Once the local convexification is performed for all *PC* cells in the finest level of detail the following coarser level is processed. As previously indicated, an incremental strategy is performed. This means that the triangles generated in previous convexifications have to be preserved.

Following the previous example of Fig. 4, the following level of detail is analyzed in Fig. 5. The objective is the convexification of each cell, but preserving the triangles generated in the previous step. In this figure the new convexification triangles together with the triangles generated in previous steps are indicated. The new local convex hull is determined by vertices {0, 1, 15, 16}.

The procedure is followed for all *PC* cells until the coarsest level of detail is processed. One obvious but interesting property of this strategy is that cell borders are attached to preserved vertices of the TIN boundary. This can be observed in the example of the Figs. 4 and 5, where vertices 0 and 16 are preserved in the different

Fig. 5 Convexification of the coarsest level



level of details. This assures that, by using this methodology, transitions between different levels of detail in contiguous cells can be handled without discontinuities.

As will be shown in the following section, the key of the incremental convexification algorithm is that the convexification triangles corresponding to the different levels of detail can be efficiently encoded in a unified representation. This compact representation can be sent to the graphics card only once, where the specific triangles corresponding to a given LOD can be efficiently reconstructed.

3 Hybrid Model Representation

As previously indicated, the objective of the algorithm is generating the adaptive tessellation required for joining multiresolution grids with a high resolution TIN. In our proposal the adaptive tessellations are mostly pre-computed and the resulting triangles can be efficiently encoded in a unified representation. This information, together with the grid and the TIN, is sent to the graphics card to generate all the adaptive tessellations required for any LOD. In the following, the additional information to be pre-processed/sent from the CPU to the graphics card is indicated.

3.1 TIN Boundary: Incremental Convexification

In our proposal the cells partially covered by the TIN (PC cells) are adaptive tessellated according to the TIN boundary information. The convexification of the TIN is performed in the CPU and the generated triangles are efficiently encoded together with the TIN boundary. The TIN boundary representation we propose includes the TIN boundary vertices information together with connectivity information. As will be shown, the connectivity values permit the rebuilding of the triangles associated with the convexification at any LOD.

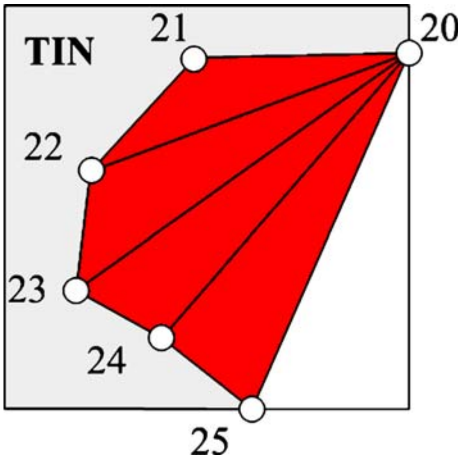
Let us represent the TIN boundary as an array of n elements $TB[i]$ with $i = 0, \cdots, n-1$ corresponding to the n vertices in the TIN boundary. Taking into account that the TIN boundary has a ring structure and following a clockwise order, consecutive vertices of the boundary are stored in consecutive positions of the array. There are two pieces of information to be stored per element/vertex: the vertex properties (coordinates, colour, ...) and the connectivity values.

The connectivity representation stores, in a very compact way, the triangles generated for all different convexification levels. For reasons of clarity, we start by working in the coarsest level of detail. As will be shown later, the representation extracted in this level can be directly applied in any other.

Taking into account the clockwise ring structure of the TIN boundary, the connectivity associated to a vertex indicates the distance (number of vertices) between the vertex and the farthest one in the ring connected to it. This way, if connectivity of vertex i is j, this means that the farthest vertex connected to it is $i + j$.

To clarify this, let us considerer the example depicted in Fig. 6. The TIN is marked and the new triangles associated with the convexification are indicated. The TIN boundary section stores the information of consecutive vertices $\{20, 21, 22, 23, 24, 25\}$. According to the above explanation the connectivities are $\{5, 1, 1, 1, 1, -\}$ i.e., vertex 20 is connected with vertex 25 (the farthest connection), while the other

Fig. 6 Connectivity structure



vertices are connected with the following in the list. We represent this information compactly by listing the vertex index and its connectivity within brackets:

$$TB = \{20(5), 21(1), 22(1), 23(1), 24(1), 25(-)\}$$

A connectivity larger than one indicates the existence of a cavity where triangles were generated. The cavity is delimited by the starting vertex and the farthest vertex connected to it. Then, if vertex i is connected with vertex $i + j$ all vertices between them are inside the cavity. We can assume that vertex i is connected to all vertices $\{i + 1, \cdots, i + j\}$, unless another nested cavity is detected. For this example, this means that vertex 20 is connected with all the following vertices $\{21, 22, 23, 24, 25\}$. Indicating this connectivity permits the construction of the triangles associated to this vertex in an easy way: $\{20, 21, 22\}$, $\{20, 22, 23\}$, $\{20, 23, 24\}$ and $\{20, 24, 25\}$. Note that once the first triangle is generated, only one additional vertex per triangle is required. This temporal exploitation of the data is very interesting for building triangles with low amount of memory accesses.

A nested cavity would be identified if one intermediate vertex has a connectivity larger than one. For example, if vertex k, with index $i < k < i + j$ has a connectivity $m > 1$, this means that there is a nested cavity between vertices k and $k + m$. The existence of this local nested cave implies that vertex i is connected to all vertices inside its cave, but not contained in the nested cave i.e., it is connected to vertices $\{i + 1, \cdots, k\}$ and $\{k + m, \cdots, i + j\}$.

A more complex connectivity structure was depicted in Fig. 5. The corresponding TB information can be summarized by the array:

$$TB = \{\ 0(1), 1(14), 2(1), 3(10), 4(9), 5(6), 6(1), 7(2), 8(1), 9(1),$$
$$10(1), 11(1), 12(1), 13(2), 14(1), 15(1), 16(1)\ \}$$

Then vertex 0 is connected with the following one (vertex 1); vertex 1 is connected with that at distance 14 i.e., vertex 15, and so on. Let us analyze vertex 4, with connectivity 9. This means that vertex 4 is connected with vertex 13, and that all vertices between them are in the cavity. These vertices are: 5(6), 6(1), 7(2), 8(1), 9(1), 10(1), 11(1), 12(1). The connectivity values indicate two nested cavities: between

5 and 11, between 7 and 9. The algorithm assumes a sequential connection of the starting vertex of a cavity to all vertices inside it but this connecting structure is broken if nested caves exist. For this example, vertex 4 is connected to all vertices between 5 and 13 but not inside a nested cavity, that is: {5, 11, 12, 13}. Vertex 5 is connected with all vertices between 6 and 11 but not inside a nested cavity, that is: {6, 7, 9, 10, 11}, and so on.

As result, the vertex connectivity can be fully specified by the utilization of an index per vertex. Until now we have verified how the triangles associated with the coarsest LOD must be reconstructed. However, as will be shown next, the method permits the reconstruction of the triangles associated with any LOD. This can be done by checking if the cavities are fully included in the cell, because larger caves are associated with coarser cells.

To clarify this, let us follow with the example of Fig. 5 and let us now consider the finer level of detail represented in Fig. 4. The TB information was generated for the coarsest level of detail but, as will be shown now, permits the reconstruction of the triangles associated with a finer level of detail.

As an example the up-right cell is analyzed. Only five vertices of the TIN boundary have to be analyzed: {0(1), 1(14), 2(1), 3(10), 4(9)}. As can be observed some of the values indicate connections with vertices outside this cell, these connections are ignored. Specifically, the connectivity values indicate connections between vertices: $0-1, 1-15, 2-3, 3-13, 4-13$. The reconstruction is as follows: Vertex 0 is connected to the following vertex 1. Vertex 1 would be connected to all vertices between 1 and 15 but vertices from 5 to 15 are not in the cell, so connectivity to vertices between 1 and 4 are assumed unless there is a nested cave. As we can see, there is a cave starting in vertex 3, so that vertex 1 has connections only with vertex 2 and 3. Vertex 2 is connected to vertex 3. Vertex 3 starts a big cave from vertex 3 and vertex 13, any case only vertex 4 is inside the cell so connection with it is assumed.

In summary, the method proposed permits the reconstruction of the convexification triangles associated with the different levels of detail in a direct way. The connectivity information required is small, as only one index per vertex has to be included. The simplicity of the tessellation procedure, as well as the efficient data management obtained, is directly related to this representation.

3.2 Correspondence Between Models

In the previous subsection we have presented the local information, required per PC cell, to reconstruct the convexification triangles associated with each level of detail. This implies the inclusion of two inherent pieces of information: on the one hand the PC cells have to be easily identified and, on the other hand, the list of TIN boundary vertices in each cell has to be indexed. In summary, this permits finding a correspondence between the two models. Additionally, once the TIN is convexified a tessellation with cell corners has to be performed. To do this, we require as additional information the list of sorted uncovered corners. In the following we propose an efficient methodology for encoding all this information.

First we choose a classification terminology for the cells. The regular grid cells are sent from the CPU selectively processed: NC cells uncovered by the TIN, can be directly rendered, CC cells, completely covered are not rendered and partially covered PC cells have to be adaptively tessellated. We associated a code to each cell to indicate its classification as NC, CC and PC. By **Grid Classification List** (GC) we denote the sequence of codes of the different cells of the regular grid for a given LOD. As usually the TIN only cover small sections of the terrain model, we employ a variable length code with the shortest code associated to the NC cells: 0 code for NC cells, 10 for CC cells and 11 for PC cells. As an example, the GC list for the four cells of Fig. 4 would be (cells organized in row order, from left to right):

$$GC = \{11, 11, 11, 0\}$$

This list indicates that the first three cells have to be adaptively tessellated while the fourth one can be directly rendered.

Those cells identified as PC have to be further processed. For the adaptive tessellation, two pieces of information are sent: the indexes of the TIN boundary vertices located on it, and the list of sorted indexes of the uncovered corners employed for the tessellations. We employ a **Vertex Classification List** (VC) for addressing the TIN boundary vertices and for listing the set of corners. In the following and for reasons of clarity, we consider the simple case with only one section of TIN boundary per cell. Generalization to larger number of sections can be found in Appendix.

Each list component, associated with each PC cell, has the following structure: $\{(A, L), (C, I)\}$. (A, L) indexes the TIN boundary vertices inside the cell. A indicates the index (in the TIN boundary array) of the first vertex of the section, while L indicates the number of consecutive vertices included in the section (consecutive positions in the TIN boundary array). On the other hand, (C, I) indicates the corner information for the tessellation. Specifically, C indicates the starting corner and I the number of uncovered corners. As an example, the vertex classification list for the three PC cells of Fig. 4 would be:

$$VC = \{(4, 9)(2, 1); (0, 5)(2, 2); (12, 5)(1, 2)\}$$

Note that the cells are listed following a row order and that the corners notation employed is: 0 (up-left), 1 (up-right), 2 (down-right) and 3 (down-left). In this example, the first block of information is associated with the up-left cell. Specifically, (4, 9) indicates that we have to read nine consecutive vertices from the TIN boundary array starting in vertex 4: {4, 5, 6, 7, 8, 9, 10, 11, and 12}. And (2, 1) indicates that the corner tessellation involves only corner 2.

4 Tessellation Algorithm

The complexity of some algorithms involved in an adaptive tessellation for hybrid representation suggests, as previously mentioned, the pre-computation and efficient encoding of the resulting triangulations. This way, only simple decoding operations and simple computations have to be performed when the model is rendered. In this section we present the main algorithm steps to be performed for the convexification reconstruction and final tessellation of the convexified TIN with the cell corners.

The algorithm involved in the reconstruction of the triangles associated with the convexification is presented next. As previously indicated, the triangulation is computed in the CPU as a pre-processing step and the information is efficiently encoded through connectivity indices. This information can be efficiently decoded during the rendering process as will be shown in this section.

```
1   $k = 0$; $count = 0$;
2   for ($v_i$ with $i = n, \cdots, n + L - 1$){
3       $count + +$;
4       if ($k = 0$)
5          Convex Hull $\leftarrow v_i$;
6       else {
7            /* Building triangles inside
8               cavities */
9            if ( $v_i.c = 1$ ){
10              if ( triangle = complete )
11                 Rendering(T);
12              else Assign_$v_i$_Triangle( );
13           }
14           /* Closing a cavity */
15           while ( $count >= S_k.c$ ){
16                  $T_1 = S_{k-1}$;
17                  $T_2 = S_k$;
18                  $T_3 = v_i$;
19                  Rendering(T);
20                  /* Eliminating starting vertex */
21                  $k - -$;
22                  /* Updating count of current
23                     starting vertex */
24                  $count = count + S_k.count$;
25                  Assign_$v_i$_Triangle( );
26                }
27              if ( $k = 0$ )
28                 Convex Hull $\leftarrow v_i$;
29            }
30            /* Starting a cavity */
31            if ( $v_i.c \neq 1$ and $(v_i.c + i) < n + L - 1$ ) {
32               /*storing previous counter */
33               $S_k.count = count$;
34               $k + +$;
35               /* Starting vertex storage */
36               $S_k = v_i$;
37               $count = 0$;
38            }
39  }
```

Fig. 7 Tessellation algorithm

Let us denote the TIN boundary vertices inside the cell as $\{v_n, \cdots, v_{n+L-1}\}$ and their connectivity values as $\{v_n.c, \cdots, v_{n+L-1}.c\}$. T_1, T_2 and T_3 denote the vertices of generated triangle T. The starting vertex of each nested cavity under construction is stored in an array called S_k with $k > 0$ and $S_k.count$ temporally stores the already processed vertices inside a cavity once a nested cavity k is opened.

The structure of the algorithm is as follows (see pseudo-code in Fig. 7): The TIN boundary vertices ($v_n \cdots v_{n+L-1}$) that fall into the cell are sequentially processed (line 2). All vertices not included in any cavity are considered part of the convex hull (lines 4 and 5). Once a cavity starts, the following vertices build the triangles in the cavity under construction (lines 7 to 13). Additionally, when a vertex closes a cavity, the starting vertex of the cavity is eliminated from the S list and an additional external triangle is identified (lines 14 to 26). Additionally, if the connectivity of the vertex is larger than one it is included in the S list as starting point of a new nested cavity (lines 30 to 38).

As an example of application let us consider again the up-left cell of the example depicted Fig. 4. The VC list associated with this cell is (4, 9)(2, 1). The TIN boundary array information associated with this cell is also replicated here for convenience:

$$TB = \{4(9), 5(6), 6(1), 7(2), 8(1), 9(1), 10(1), 11(1), 12(1)\}$$

The vertices are sequentially processed and the associated triangles are identified. In Table 1 we briefly indicate the main steps of the tessellation algorithm: the columns represent the vertex to be processed, the current starting vertex under analysis, the triangle under construction and the S list status.

In this example, vertex 4, with connectivity larger than 1, is a starting vertex. After this, vertex 5 is read and considered to be the new starting vertex and so a nested cavity is constructed. The following vertices, 6 and 7, permit the construction of the

Table 1 The tessellation procedure for the example depicted in Fig. 4

Vertex	Starting vertex	Triangle	*S* list
4(9)	4	(4, ,)	{4} ← 4
5(6)	5	(5, ,)	{4, 5} ← 5
6(1)	5	(5, 6,)	{4, 5}
7(2)	7	(5, 6, 7)	{4, 5}
	7	(7, ,)	{4, 5, 7} ← 7
8(1)	7	(7, 8,)	{4, 5, 7}
9(1)	7	(7, 8, 9)	{4, 5, 7} → 7
	5	(5, 7, 9)	{4, 5}
10(1)	5	(5, 9, 10)	{4, 5}
11(1)	5	(5, 10, 11)	{4} → 5
	4	(4, 5, 11)	{4}
12(1)	4	(4, 11, 12)	{ } → 4

first triangle (5, 6, 7). Vertex 7 starts a new nested cavity that ends in vertex 9. Then, when triangle (7, 8, 9) is built, the cavity is closed and vertex 7 is eliminated from the *S* list. Elimination of vertex 7 implies the generation of the external triangle (5, 7, 9) involving the new current starting, vertex 5. The procedure follows until all vertices in the cell are processed. Note that the connectivity of vertex 4 indicates that it is connected to vertex 13, a vertex not included in the cell. This way, convexification triangles corresponding to higher levels of detail cannot be generated, because they involve vertices that fall outside the current cell.

This procedure permits the reconstruction of the triangles associated with the convexification algorithm. As can be derived from the code algorithm, the computational requirements are low, as only additions/comparisons are involved.

Finally, the corner tessellation procedure receives the TIN boundary vertices of the convex hull (after convexification procedure) together with the sorted corners for the tessellation procedure (information contained in the VC list). As was explained in Section 2.1, this permits a simple tessellation procedure in which the set of sorted corners and TIN boundary vertices are sequentially connected. This procedure permits the reconstruction of the triangles associated with the convex hull of the TIN boundary and the cell corners. The computational requirements of the algorithm are very low, as they are mainly associated with the corner shift control. This control can be implemented with the sign evaluation of simple cross products.

5 Hybrid Extension with Sub-cell Generation

In this section we present an extension of the previous methodology for hybrid representations. This extension generates a hybrid structure per cell with sub-cells and triangles and the only additional requirement is the pre-computation of a tree storing a sub-cell hierarchy. As a result, with only this additional information, the two different tessellation procedures can be supported.

The new procedure is based on the hierarchical/recursive subdivision of partially covered cells into sub-cells. This procedure is repeated until the unit cell size is achieved (size of the finest LOD resolution). To complete the procedure, the unit PC sub-cells are tessellated into triangles. The triangles generated represent the area uncovered by the TIN and connect the sub-cell corners with the TIN boundary vertices.

The objective of this combined system is to preserve the original grid information where possible, adapting only the structures that are in close proximity to the TIN boundary. This permits the exploitation of the information offered by both representations, limiting the areas of interconnection and manipulation of the original data.

To clarify this, let us consider the example of Fig. 8a. In this figure a PC cell of the grid and the TIN silhouette inside it are indicated. Let us label the current LOD of the grid representation as LOD 2 and let us assume that there are only two finer resolution LODs for the grid (LOD 1 and LOD 0). This means that the cell has to be recursively subdivided two times until the unit cell size (LOD 0) is achieved. In the first subdivision procedure (Fig. 8b) two *PC* and two *NC* sub-cells are identified. The *NC* sub-cells can be directly rendered while the *PC* sub-cells are subdivided again. As result (see Fig. 8c), three *NC* and five *PC* unit size sub-cells are additionally generated. The *NC* sub-cells are sent to the graphics pipeline while the *PC* unit sub-cells are triangulated. This is indicated in Fig. 8d.

The tessellation of each unit size sub-cell is based on the TIN convexification and the corner tessellation. This means that by extending the previously presented methodology to this new tessellation procedure, it can be directly performed if a procedure for sub-cell generation is included. Due to the complexity of the algorithm involved and following the previous strategy, in our proposal this procedure is performed as a pre-processing step and the result is efficiently encoded.

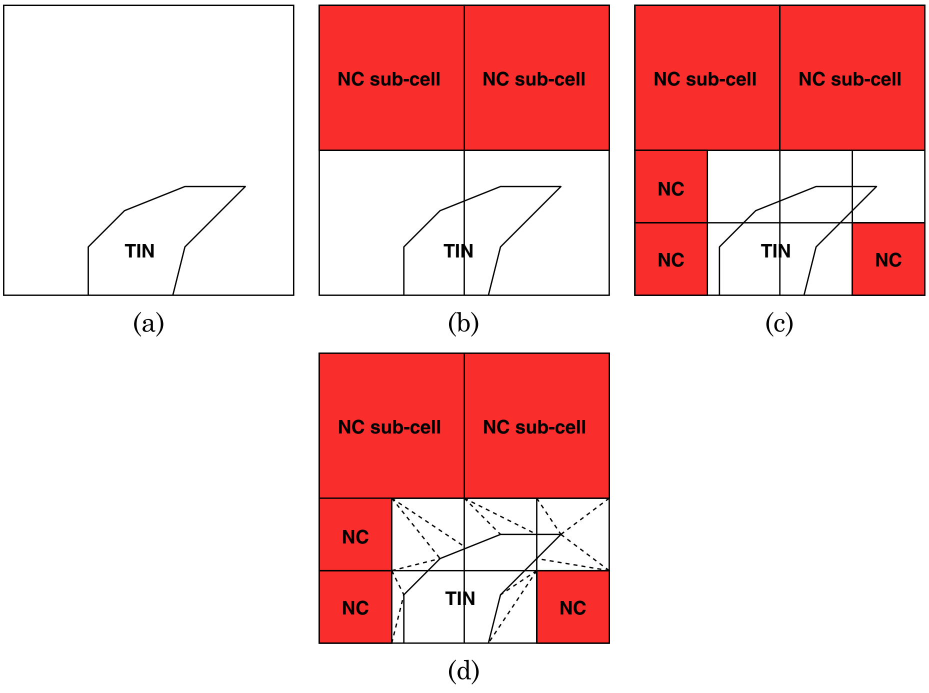


Fig. 8 PC subdivision example for the hybrid algorithm

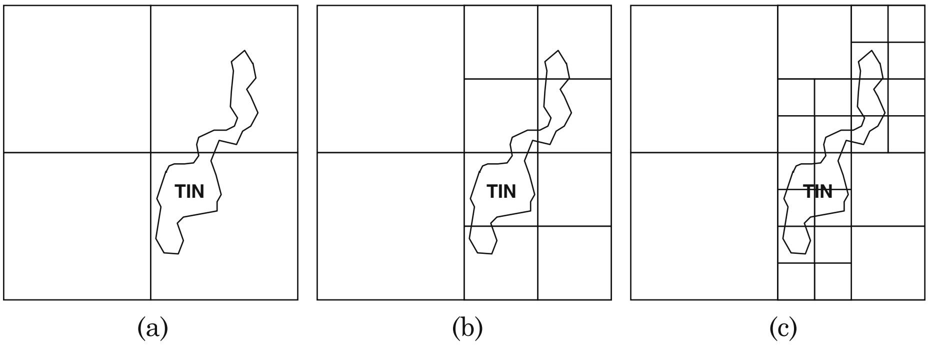


Fig. 9 GRID subdivision example for the hybrid algorithm

5.1 Sub-cell Structure: Tree Representation

The recursive subdivision to be performed in each *PC* cell can be efficiently stored in a tree representation. As an example let us consider the system of four cells and a TIN structure depicted in Fig. 9a. The *GC* list associated with this system is 0, 11, 0, 11 where the cells are sequentially listed following a row order. As was previously indicated, each *PC* sub-cell is subdivided into four sub-cells and the procedure is recursively repeated until the finest resolution. Let us assume in this example that there are two finer LODs. The result of the first step of subdivision is depicted in Fig. 9b. That is, the up-right cell is subdivided into four sub-cells that are classified as (*NC*, *PC*, *PC*, *PC*) i.e., (0, 11, 11, 11), and the right-down cell is subdivided into four sub-cells (*PC*, *NC*, *PC*, *NC*), that is (11, 0, 11, 0). Each *PC* sub-cell is subdivided again obtaining the structure of Fig. 9c.

This structure can be efficiently represented in a tree structure. Each node of the tree represents a cell/sub-cell and each row of the tree a step in the subdivision process. For this example the tree is depicted in Fig. 10. This tree can be schematically represented by the sequence of nodes in each row. Specifically for this example:

$$\begin{aligned} \mathit{Tree} = {} & (0, 11, 0, 11)(0, 11, 11, 11)(11, 0, 11, 0)(0, 0, 11, 0)(0, 0, 0, 11) \\ & (11, 0, 11, 0)(11, 11, 11, 0)(11, 0, 0, 0) \end{aligned}$$

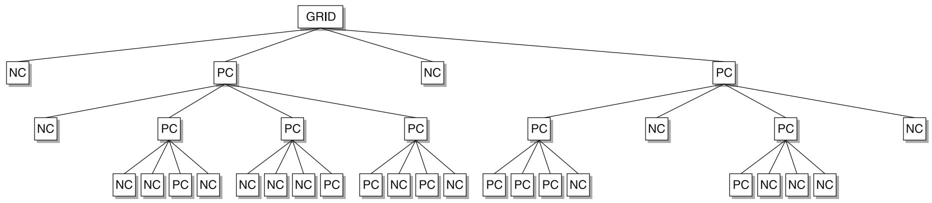


Fig. 10 Tree representation example

This tree contains all the information required for the subdivision of each cell into sub-cells. This information, together with the local information for local convexification and corner tessellation of each unit size sub-cell permits the hybrid representation.

5.2 Tessellation algorithm

The tree storing the sub-cell structure is generated as pre-processing step. This information is applied during the rendering process to identify the *PC* cells, to subdivide them into *CC* sub-cells (to be eliminated), *NC* sub-cells (to be sent to the graphics pipeline) and *PC* unit sub-cells. The *PC* sub-cells are further subdivided. The process is repeated until the finest resolution level is reached. The final *PC* unit sub-cells are tessellated according to the convexification and corner tessellation procedure previously presented.

Each cell subdivision implies the computation of the coordinates of five new points. To avoid discontinuities between cells, simple linear interpolations are performed to generate the new sub-cells corners. This implies that the subdivision procedure only implies decoding the tree information and linear interpolation operations. Once the unit-size *PC* sub-cells are generated and identified, the convexification and corner tessellation procedure are performed (see Section 4).

Both tessellation algorithms share the same core structure (convexification and corner tessellation). As result, both algorithms can be supported with only small additional requirements. This permits the operation of a framework in which the user can select between hybrid tessellations in function of visual quality criteria.

6 Results

The methodology we propose permits the hybrid representation of terrains by performing adaptive tessellation in the contour of the TIN. The algorithms proposed permit the generation of the adequate tessellation structure for any level of detail. The algorithm is based on a mixed strategy where part of the information is pre-computed and only simple operations are performed ad hoc. Due to the simplicity of the computations to be performed, hardware support can be included in the graphics card for performing the final operations. As a result, the user can choose between two tessellation procedures in real time following quality criteria. In the following, we analyze the properties of our proposals in terms of the communication requirements and final image quality obtained.

The communication/storage requirements are strongly dependent on the application. The size and structure of the TIN determines these requirements and due to this dependence, giving generic estimations is far from being realistic. In the following we develop an analysis of these requirements giving specific results for a set of test models.

We start by analyzing the first algorithm based on triangles generation. In this tessellation algorithm, the convexification and corner tessellation procedures require the TIN boundary (vertices and connections) and the *GC* and *VC* lists. The TIN boundary information is independent of the level of detail selected so that the information can be sent only once and re-employed in dynamic applications. The storage

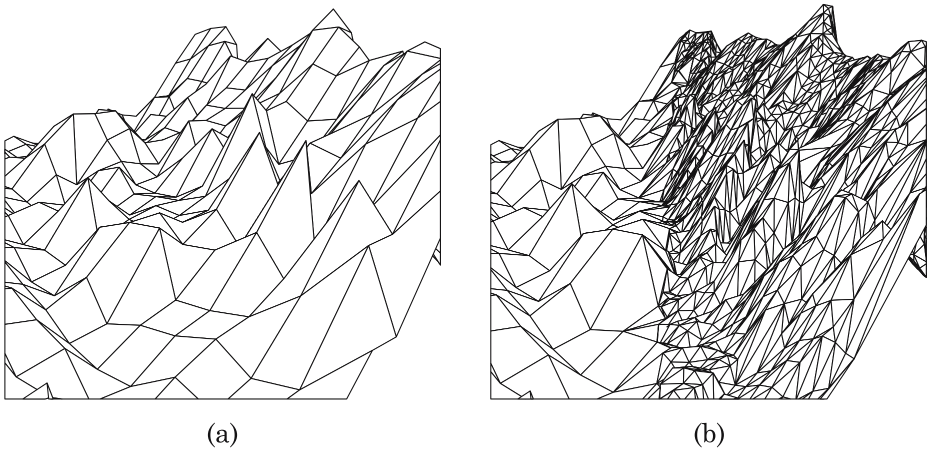


Fig. 11 Model 1 **a** Grid **b** Final model

requirements depend on the number of vertices on the boundary of the TIN and the connectivity structure between them. The connectivity index range is determined by the maximum number of TIN boundary vertices per cell. Assuming a maximum number of vertices L_{max}, the connectivity index associated with each TIN boundary vertex would require $log_2(L_{max})$ bits. For example, for up to $L_{max} = 256$ (larger number in practical situations) only 8 additional bits per TIN boundary vertex are required.

With respect to the *GC*, this includes the information to identify each cell as *PC*, *CC* or *NC*. As was previously indicated, a variable length code of up to two bits per cell was employed.

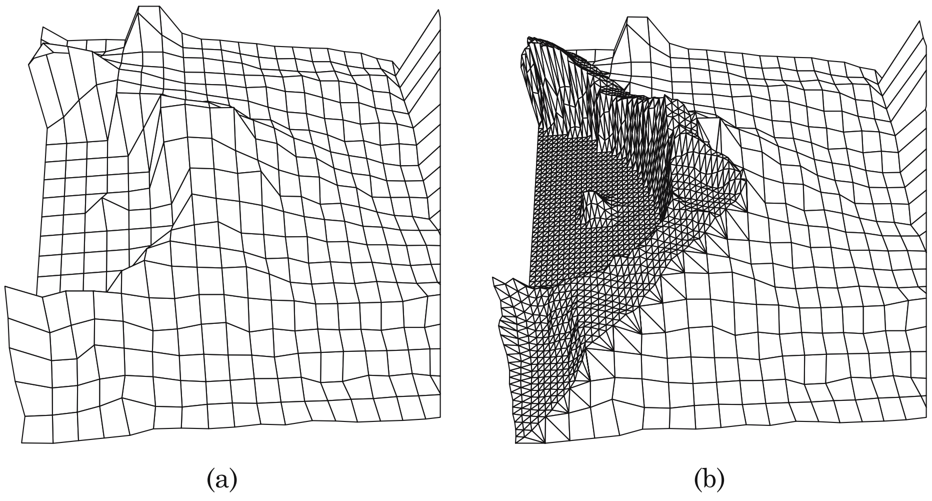


Fig. 12 Model 2 **a** Grid **b** Final model

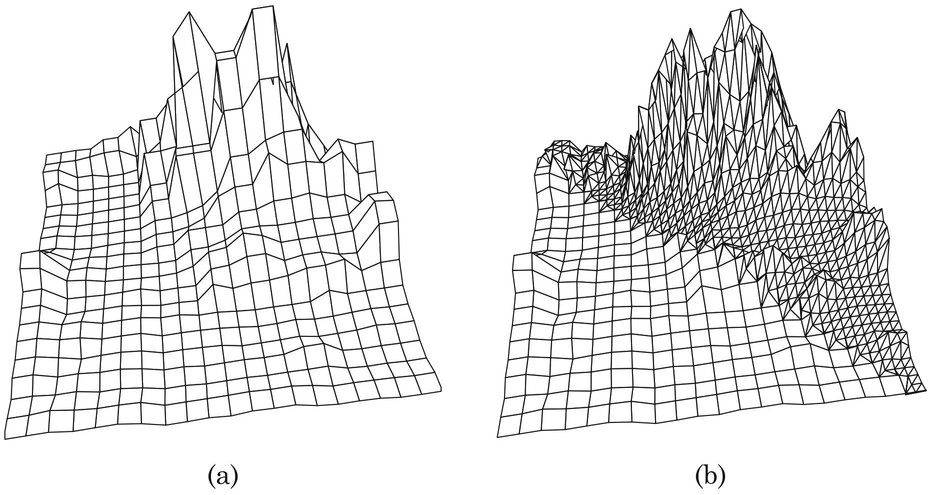


Fig. 13 Model 3 **a** Grid **b** Final model

The VC list stores additional information required per PC cell. This information includes two fields, one for the addressing of the TIN boundary array (A, L), and the other for indicating the corners involved in the tessellation (C, I). The A value is determined by the number of vertices in the TIN boundary and depends directly on the application example. If for example we work with a TIN boundary with 65 k vertices we require 16 bits to address a specific vertex. On the other hand, $log_2(L_{max})$ bits are required for encoding the section length. For example, continuing the previous example, if $L_{max} = 256$ vertices only 8 bits are required. With respect to the corner information, 4 bits are required for encoding the C and I fields. In summary, for this example 24 bits per PC cell would be required for the VC list. These requirements

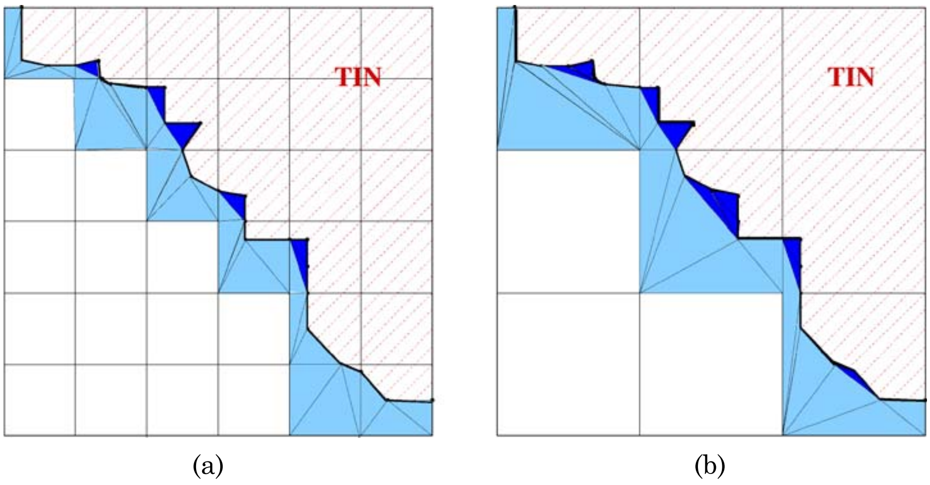


Fig. 14 Detailed view of Fig. 11: **a** Finer LOD **b** Coarser LOD

are small if we take into account that 32 bytes[1] are required for a vertex in a standard triangular mesh.

Giving generic communication requirements in practical situations is very difficult as it strongly depends on the cartographic area to be represented. As an example we list here the results associated with three models, called Model 1, 2 and 3, respectively. In the three cases the model consists of a regular grid partially covered by a TIN. These models are depicted in Figs. 11, 12 and 13. Specifically, Figs. 11a, 12a and 13a represent the regular grids while Figs. 11b, 12b and 13b represent the final models (grid plus TIN). The benefits in terms of quality of including the TIN models are obvious and, as expected, no discontinuities are generated. Note that the final models represented in these figures include triangles associated to the TIN, to the convexification procedure and to the corner tessellation algorithm. This different triangle origin is emphasized in Fig. 14 where a 2D view of two consecutive levels of detail corresponding to a small area of Fig. 11 are depicted. In this figure convexification triangles are indicated in dark colour while corner triangles in light colour.

The number of vertices of the TIN boundary of the three models employed are 267, 248 and 239, respectively. Although the area covered by the TIN is large, the number of TIN boundary vertices is still low (less than 300 vertices). Table 2 summarizes the storage/communication requirements obtained for different LODs where LOD 0 represents the finest level of detail and LOD 2 the coarsest one. The largest TIN boundary employed is compounded of up to 267 vertices and the maximum connectivity obtained for the three models is 19. The communication requirements per *CC* or *NC* cells are 2 and 1 bits, respectively. The percent of *PC* cells (see the fourth column) in these models is lower than 10.5% in the finest LOD and 36% in the coarsest LOD. For the *PC* cells the section codification has to be included (A, L, C, I values). A can be represented by 9 bits (Model 1) and 8 bits (Model 2 and 3). The L_{max} value is represented in the fifth column of Table 2 for each model. Besides this, the number of bits required to represent C and I are 2 bits each. This results in a low number of bits per section (last column of the table).

Table 2 Storage/communication requirements for the different models

		Connectivity max.	PC	L_{max}	Bits/section
Model 1	LOD 0	8	42 (10.5%)	7	16
	LOD 1	4	19 (19%)	17	18
	LOD 2	4	9 (36%)	37	19
Model 2	LOD 0	4	38 (9.5%)	7	15
	LOD 1	6	18 (18%)	17	17
	LOD 2	8	9 (36%)	37	18
Model 3	LOD 0	2	39 (9.75%)	5	15
	LOD 1	7	19 (19%)	11	16
	LOD 2	19	9 (3%)	23	17

[1]We consider every vertex as compounded by its position (3 coordinates), its normal vector (3 coordinates) and a texture value (2 coordinates); and for the encoding of every component of the position, normal texture, we assume a standard word size of 32 bits.

With respect to the second algorithm, a tree for storing the sub-cell structure has to be included. To estimate the communication requirements associated with this tree, let us consider a regular grid with N LODs. The subdivision of one PC cell into sub-cells could generate up to 4^i PC sub-cells in each subdivision step ($i = 0, \cdots, N-1$). As 2 bits are required per sub-cell, the communication requirements for the PC cells of the coarsest regular grid is up to: $\sum_{i=0}^{N-1} 2 \cdot 4^i$.

For example, for $N = 3$ LODs, the maximum number of bits per PC cell for the tree representation would be 42 bits. This in an upper limit representing the situation in which each PC sub-cell is subdivided recursively into four PC sub-cells. The requirements associated with the tree grow exponentially with the number of LODs. However, due to the low number of PC cells and PC sub-cells these requirements are low in practical situations.

The adaptive tessellation generated for both algorithms is different, as can be seen in the models represented in Fig. 15. Three levels of detail were depicted for both algorithms, from finest (left) to coarsest (right). The first algorithm generates less primitives for rendering, while the second one reduces the area of triangle generation to the locality of the TIN contour. For example, for the coarsest level of detail the number of triangles generated with the first algorithm is about 50% the number of primitives generated by the second algorithm. Note, that a reduced number of primitives for coarser levels of detail is desired, as this corresponds to areas that will be depicted with low resolution. Although the quality could be better for the second algorithm, at very coarse levels of detail this quality is not required and, moreover, the application would probably not represent the TIN.

With respect to the quality of the models generated, the benefits of our strategy are clear taking into account the increment in quality when the DEM representation is enriched with the available TIN meshes representing detailed areas of the terrain (see as an example Figs. 11, 12 and 13). Although hybrid representation of terrains

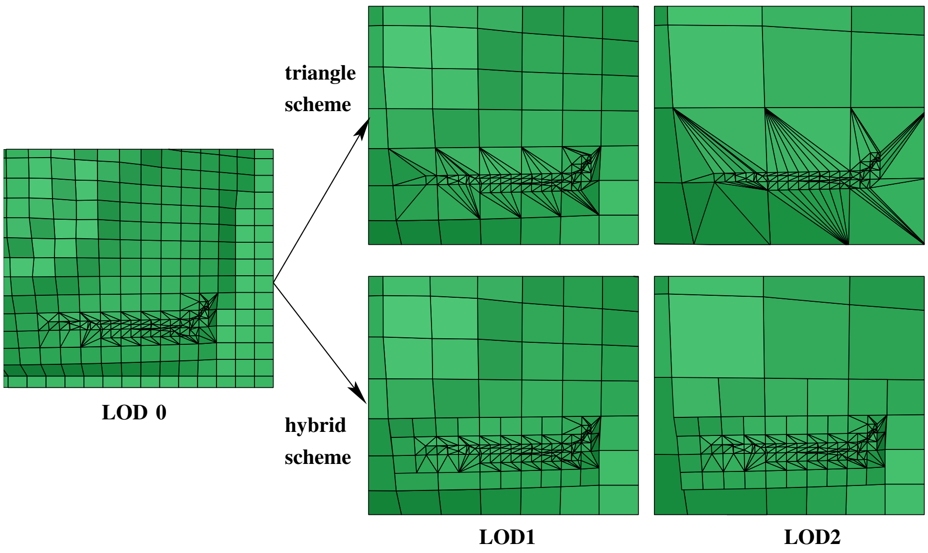


Fig. 15 Adaptive tessellation example for both schemes

combining different data sets is of great interest, the traditional solution is the disconnected representation of the data sets or the pre-processing in software of the combined structure. These strategies are not appropriate when quality or real time rendering are the objectives. We did not find any proposal in the bibliography that attempts to solve the challenge of real time rendering of a multiresolution hybrid representation composed by TINs and DEMs. In this sense, no comparisons in terms of quality with other proposals can be performed.

Note that our presentation is focussed on the combination of multiresolution grids with a high resolution TIN. However, extension to a framework including also a multiresolution TIN system would be straight forward. Then, when a zoom out is performed the high resolution TIN would not be required and a lower resolution model could be employed. Following our proposal and to reduce the information to be sent to the GPU, one TIN resolution level would be associated to multiple consecutive grid resolution levels. Then, our solution could be applied to encode the correspondence between each TIN level with the associated grid levels.

To include more complex data structures in our representation, we have extended our algorithm to manage situations with multiple sections of the TIN boundary per grid cell (see Appendix). We have tested our algorithm with different irregular models obtaining in all cases final meshes without discontinuities. According to our tests and analysis we conclude that our solution permits the process of very irregular systems generating final models without discontinuities.

Improving the quality of the resulting models following our strategy would imply, as first challenge, changing the corner tessellation algorithm to produce a different tessellation structure to join the models. Our corner tessellation algorithm is characterized by its simplicity, the correctness of the resulting tessellations and the sequential structure of the algorithm. Due to this sequential structure, the triangles generated do not require a posterior correction, avoiding irregular algorithms with a forward–backward structure. However, the triangles generated with this method can be, in some situations, long and narrow and a different tessellation could be desirable. For this reason, we are currently working in other tessellation algorithms that could improve the quality of the tessellation while keeping its simplicity, correctness and sequential structure.

In summary, as previously mentioned, high quality models with very similar communication and computation requirements for both algorithms are obtained. Due to the simplicity of both techniques, hardware support can be included to perform the tessellations in hardware. This way, the user could switch between models in real time, in order to choose the most adequate tessellation according to visual quality decisions.

7 Conclusions

We have developed a new technique for hybrid representation of terrain models, where a multiresolution grid together with a high resolution TIN can be combined into a unified model. Specifically, we have developed an adaptive tessellation technique based on an efficient representation of the tessellation structures required for joining the models.

Our proposal permits the adaptive tessellation of each grid cell partially covered by the TIN based on the TIN boundary information. High quality rendering is obtained, as no discontinuities are generated between representations. The method is based on the efficient representation of the information to be sent from the CPU to the graphics pipeline. The representation we propose can be easily extended to include different tessellation procedures. The operations to be performed to decode and complete the tessellation structure are very simple.

Acknowledgements This work was partially supported by the Ministry of Science and Technology of Spain under contract MCYT TIN2004-07797-C02 and by the Secretaría Xeral I+D of Galicia (Spain) under contract PGIDIT03TIC10502PR.

Appendix

Extension to Complex TIN Structures

In this section we generalize our representation methodology to include those situations with multiple TIN boundary sections falling into the same grid cell. This generalization permits the extension of our method for representing any irregular TIN structure.

When two or more sections fall into the same cell, the convex hull is composed by sets of vertices, each set with vertices from two consecutive TIN sections. Specifically, for N sections we will define N sets of convex hull vertices, where the ith, $0 \leq i \leq N-1$, set of vertices of the convex hull is composed by the initial vertices of the ith section and the last vertices of the (i-1)-th section. In Fig. 16a an example with two TIN boundary sections is depicted. Both TIN boundary sections are labelled following a clockwise order as explained before. The first TIN is composed by vertices {1,2,...,13} and the second one by {22,...,28}. In Fig. 16b the convexification structure is depicted. This convex hull is composed by the following sequence of TIN boundary vertices: {1,2,6,7,27,28} in the upper part and {22,11,12,13} in the lower part, both sets composed by two sequences of vertices corresponding to the two TIN boundary sections. This way, the upper part of the convex hull is composed by the initial vertices of the first TIN {1,2,3,6,7} (note the local caves are identified in this procedure) and the last

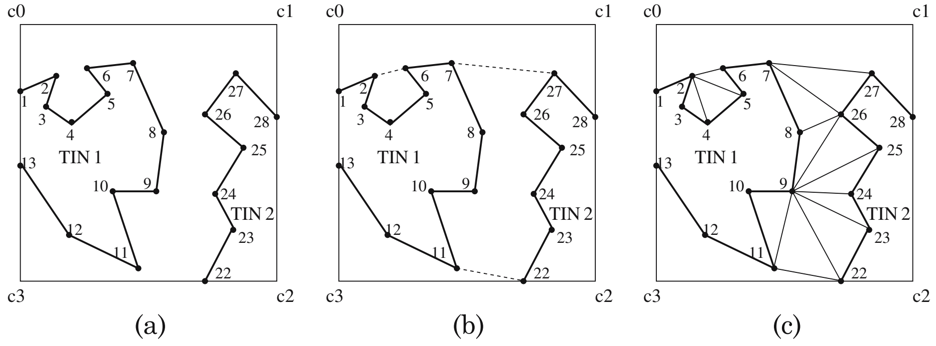


Fig. 16 Example with two sections. **a** TIN information **b** Convex hull **c** Convexification triangles

vertices of the second TIN {27,28}. On the other hand, the lower part of the convex hull is composed by the first part of the second TIN {22} and the last part of the first TIN {11,12,13}.

In the simple case of one TIN section per cell the sequence of convex hull vertices are consecutively connected with a group of corners. In case of multiple TIN sections per cell, each set of vertices is connected with a given group of corners. Following with the example of Fig. 16 this means that the upper part of the convex hull is joined with corners {0,1} while the lower part with corner {3}.

To preserve the structure of our algorithm, we propose to apply the same method to generate the convexification triangles and to identify the convex hull vertices. To cover those situations with multiple sections only some additional control and a modified VC list have to be employed. The additional control is included to detect the convex hull vertices that start/close the multi-section caves. This can be performed by analyzing the connectivity information associated to the TIN boundary vertices. As an example, Fig. 16c shows a possible configuration of convexification triangles associated to our system with two TIN sections. In this case the TIN boundary information would be:

$$\begin{aligned} TB(1) &= \{..., 1(1), 2(4), 3(1), 4(1), 5(1), 6(1), 7(20), 8(18), 9(17), 10(1),\\ &\quad 11(11), 12(1), 13(1), ...\} \\ TB(2) &= \{..., 22(1), 23(1), 24(1), 25(1), 26(1), 27(1), ...\} \end{aligned}$$

Some additional control has to be included to detect that a multi-section cave starts in vertex 7 and ends in vertex 11. The control can be implemented in a simple way by detecting the first (vertex 7) and last (vertex 11) vertices connected with other TIN section.

To complete the representation we have to extend the vertex classification list to take into account the multiple origin of the convex hull and the distribution of corners in groups. Then, vertex classification list has to be generalized and each partial covered cell would be represented by:

$$\{N; (A^0, L^0)(C^0, I^0); (A^1, L^1)(C^1, I^1); ...; (A^{N-1}, L^{N-1})(C^{N-1}, I^{N-1})\} \qquad (1)$$

where N indicates the number of TIN boundary sections falling into the cell, (A^i, L^i) describes the i-th section, $0 \le i \le N-1$, and (C^i, I^i) indicates the corner information to be considerer for corner tessellation of the ith set of convex hull vertices. As an example, the VC list for Fig. 16 is:

$$VC = \{2; (1, 13)(0, 2); (22, 7)(3, 1)\}$$

That is, we have two sections, first one with 13 vertices and starting in vertex 1, and second one with seven vertices starting in vertex 22. The first set of convex hull vertices implies corners 0 and 1, while the second set of convex hull vertices will be connected to corner 3.

Another interesting example with two TIN boundary sections is indicated in Fig. 17a. The boundary vertices are {240,241,242} and {246,247,248} and, in this example, the four corners of the grid cell are completely covered by the TIN. The

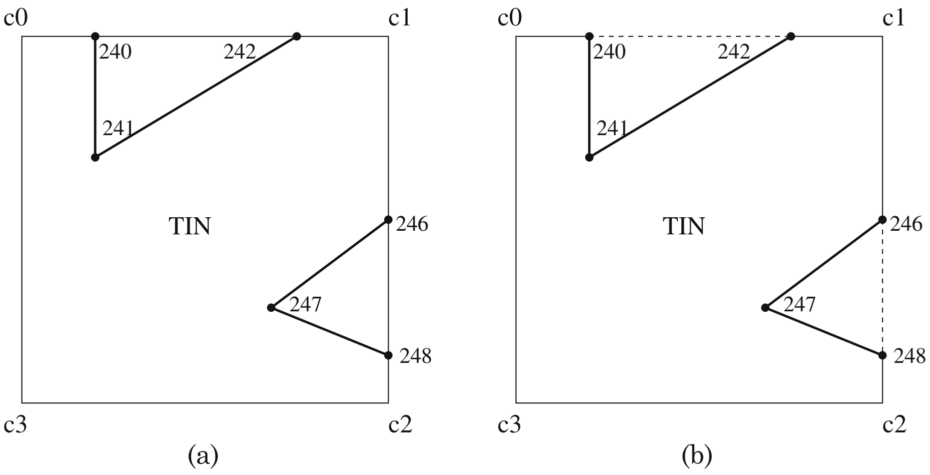


Fig. 17 Example with two sections and the corners fully covered. **a** TIN information **b** Convex hull

result of the convexification procedure is depicted with dashed lines in Fig. 17b. Note, that as result of the convexification procedure two new convexification triangles are generated and no corner tessellation procedure is required. This cell would be described through the following VC list:

$$VC = \{2; (240, 3)(0, 0); (246, 3)(0, 0)\}$$

where the corner list can be specified with any code $(i, 0)$, $0 \leq i \leq 3$, as no corner is employed for tessellation.

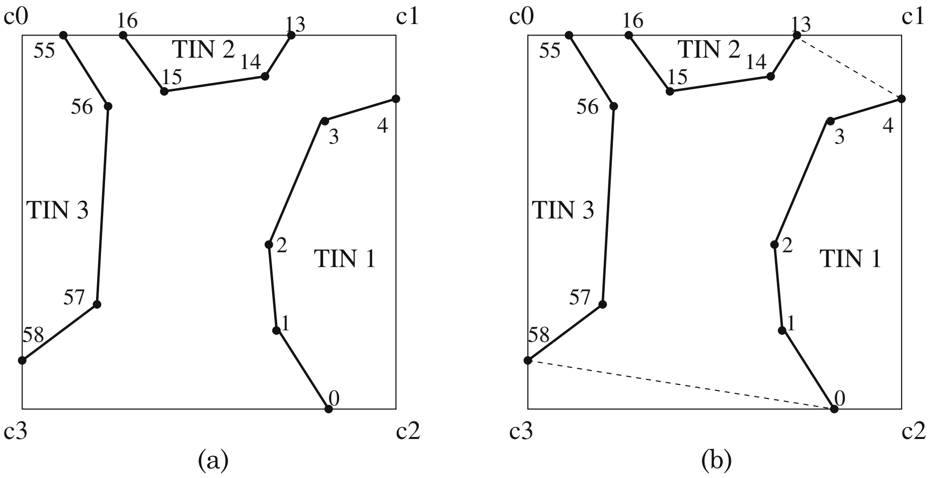


Fig. 18 Example with three sections. **a** TIN information **b** Convex hull

Finally, we can analyze the example of Fig. 18a where three TIN boundary sections are detected. In this case the VC list would be:

$$VC = \{3; (0, 5)(3, 1); (13, 4)(1, 1); (55, 4)(0, 0)\}$$

where the three TIN boundary sections are defined by vertices {0,1,2,3,4}, {13,14,15,16} and {55,56,57,58}, respectively. As can be observed in Fig. 18b the first convex hull set is defined by vertices {0,58}, that is, first vertex of the first TIN boundary section and last vertex of last TIN boundary section. Corner 3 is employed for the corner tessellation in this case. The second convex hull set is defined by vertices {13,4}, that is, first vertex of second section and last vertex of first section. The corner to be employed for tessellation is corner 1. The third convex hull set is defined by {55,16}, that is, first vertex of third section and last vertex of second section. No corner is required for tessellation.

References

1. M. Amor, M. Bóo, and J. Döllner. "Hardware support for hybrid grid representation of terrains," Technical Report, University of Santiago de Compostela, 2004. (www.ac.usc.es)
2. M. Amor, M. Bóo, J. Hirche, M. Doggett, and W. Strasser. "A meshing scheme for efficient hardware implementation of butterfly subdivision surfaces using displacement mapping," *IEEE Computer Graphics and Applications*, Vol. 25(2):46–59, 2005.
3. K. Baumann, J. Döllner, and K. Hinrichs. "Integrated multiresolution geometry and texture models for terrain visualization," in R. van Liere, and W. de Leeuw (Eds.), *Proc. of Joint Eurographics—IEEE TVCG Symposium on Visualization 2000*, pp. 157–166, Springer, Berlin Heidelberg New York, 2000.
4. M. Bóo, M. Amor, M. Doggett, J. Hirche, and W. Strasser. "Hardware support for adaptive subdivision surface rendering," in *Proc. of Siggraph/Eurographics Hardware Workshop*, pp. 33–40, ACM, New York, 2001.
5. P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. "BDAM—Batched dynamic adaptive meshes for high performance terrain visualization," in P. Brunet and D. Fellner (Eds.), *Proc. of Eurographics 2003*, *Computer Graphics Forum,* Vol. 22(3), pp. 505–514, Blackwell, Oxford, UK, 2003.
6. M. Doggett and J. Hirche. "Adaptive view dependent tessellation of displacement maps," in *Proc. of Siggraph/Eurographics Hardware Workshop*, pp. 59–66, ACM, New York, 2000.
7. M. Kraus and T. Ertl. "Simplification of nonconvex tetrahedral meshes," in *Electronic Proc. of NSF/DoE Lake Tahoe Workshop for Scientific Visualization*, 2000.
8. M. Kraus and T. Ertl. "Cell projection of cyclic meshes," in T. Ertl, K. Joy, and A. Varshney (Eds.), *Proc. of IEEE Visualization 2001*, pp. 215–222, IEEE Computer Society, Los Alamitos, CA, 2001.
9. P. Lindstrom and V. Pascucci. "Visualization of large terrains made easy," in T. Ertl, K. Joy, and A. Varshney (Eds.), *Proc. of IEEE Visualization 2001*, pp. 363–370. IEEE Computer Society, Los Alamitos, CA, 2001.
10. P. Lindstrom and V. Pascucci. "Terrain simplification simplified: A general framework for view-dependent out-of-core visualization," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8(3):239–254, 2002.
11. F. Losasso and H. Hoppe. "Geometry clipmaps: Terrain rendering using nested regular grids," in *Proc. of ACM SIGGRAPH 2004*, pp. 769–776, ACM, New York, 2004.
12. D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. Level of Detail for 3D Graphics. Morgan Kaufmann, San Mateo, CA, 2002.
13. J. O'Rourke. *Computational Geometry in C*. 2nd edition, Cambrige University Press: Cambridge, MA, 1998.
14. R. Pajarola, M. Antonijuan, and R. Lario. "QuadTIN: Quadtree based triangulated irregular networks," in R. J. Moorhead, M. Gross, and K. I. Joy (Eds.), *Proc. of IEEE Visualization 2002*, pp. 395–402, IEEE Computer Society, Los Alamitos, CA, 2002.

15. J. Schneider and R. Westermann. "GPU-Friendly high-quality terrain rendering," *Journal of WSCG*, Vol. 14(1-3):49–56, 2006.
16. P.L. Williams. "Visibility ordering meshed polyhedra," *ACM Transactions on Graphics*, Vol. 11(2):103–126, 1992.
17. T. Yýlmaz, U. Güdükbay, and V. Akman. *Modeling and Visualization of Complex Geometric Environments. In Geometric Modeling: Techniques, Applications, Systems and Tools (Chapter 1)*. Kluwer, Norwell, 2004.

Montserrat Bóo received the B.S. and Ph.D. degree in Physics from the University of Santiago de Compostela (Spain) in 1993 and 1997, respectively. Currently she is Associate Professor in the Department of Electronics and Computer Eng. at the University of Santiago de Compostela. Her interests are in the area of VLSI digital signal and image processing, computer graphics and computer arithmetic.

Margarita Amor is currently an associate professor at Dept. of Electronic and Systems of the Univ. of A Coruña. She received the B.S. and Ph.D. degree in Physics from the University of Santiago de Compostela (Spain) in 1993 and 1997, respectively. Her research interests is mainly focused in the areas of video processing, computer graphics and parallel computing.

Jürgen Döllner is working as professor in computer science at the Hasso–Plattner-Institute, University of Potsdam, Germany, where he directs the computer graphics and visualization division. His research interests include real-time 3D rendering, nonphotorealistic rendering, spatial visualization, and software visualization. Döllner studied mathematics and computer science, and received a Ph.D. in computer science from the University of Münster.